Personnel Scheduling as Satisfiability Modulo Theories

Christoph Erkinger and Nysret Musliu
Institute of Information Systems, TU Wien, Austria
christoph.erkinger@gmail.com, musliu@dbai.tuwien.ac.at

Abstract

Rotating workforce scheduling (RWS) is an important real-life personnel rostering problem that appears in a large number of different business areas. In this paper, we propose a new exact approach to RWS that exploits the recent advances on *Satisfiability Modulo Theories* (SMT). While solving can be automated by using a number of so-called SMT-solvers, the most challenging task is to find an efficient formulation of the problem in first-order logic. We propose two new modeling techniques for RWS that encode the problem using formulas over different background theories. The first encoding provides an elegant approach based on linear integer arithmetic. Furthermore, we developed a new formulation based on bitvectors in order to achieve a more compact representation of the constraints and a reduced number of variables.

These two modeling approaches were experimentally evaluated on benchmark instances from literature using different state-of-the-art SMT-solvers. Compared to other exact methods, the results of this approach showed a notable increase in the number of found solutions.

1 Introduction

For companies and organizations in a broad range of business areas, it is necessary to provide availability 24 hours a day and 7 days a week. While some companies need their machines up and running all the time in order to fit the production demand, other organizations like hospitals or police stations need their staff to be working day and night to ensure proper health-care or the citizens' safety.

In this paper the particular problem of interest is *Rotating Workforce Scheduling* (RWS), which is a constraint satisfaction problem (CSP). This problem addresses designing a schedule where a certain number of shifts are assigned to work days over a specific planning period, typically one week. The work schedule is designed for each employee by considering the necessary temporal requirements and other constraints. Furthermore, in rotating workforce scheduling all employees work the same schedule in a rotating sequence starting at different offsets.

Over the years different approaches to this problem have been introduced in the literature. Early methods were based on exhaustive enumeration [Butler, 1978] and Integer Linear Programming [Laporte *et al.*, 1980]. A network flow model was developed in [Balakrishnan and Wong, 1990] to solve the problem. In [Laporte, 1999] a relaxation of the constraints was proposed that allows the creation of efficient rotating schedules by hand. The use of an algebraic computational approach has been proposed in [Falcón *et al.*, 2016]. Heuristic techniques were successfully used to create rotating schedules in [Musliu, 2005; 2006]. Methods based on constraint programming techniques were applied in [Laporte and Pesant, 2004; Musliu *et al.*, 2002; Triska and Musliu, 2011]. Although these exact approaches provide a solution for many instances, solving large problems is still a challenge for them. Therefore, it is intriguing to find other exact methods that may provide better results.

SMT-solving offers the expressive power of first-order theories combined with the good performance of state-of-the-art SAT-solvers. As a consequence, researchers have gained interest in using SMT for solving constraint satisfaction problems. In [Bofill *et al.*, 2010] it is stated that SMT-Solvers are well suited for solving CSPs and a tool is presented for translating instances of the *MiniZinc* constraint modeling language into the *SMT-LIB* standard. SMT-solvers have been used to solve different constraint optimization problems [Ansótegui *et al.*, 2011], [Nieuwenhuis and Oliveras, 2006], [Ansótegui *et al.*, 2013]. In [Ansótegui *et al.*, 2013] a SMT based approach was presented for the Nurse Rosterin Problem (NRP), which is related to our problem. However, NRP differs from our problem as it does not take into consideration the cyclicity of the schedule.

In this paper we investigate how the RWS problem can be encoded in first-order formulas over certain background theories. By formulating the problem in the theory of quantifier-free linear integer arithmetic, many instances could be solved. However, this approach reached its limits when applied to problems with a large number of employees. To overcome this issue we developed a second new encoding using the theory of fixed-size bitvectors. By means of machine arithmetic we were able to achieve a very compact representation of the constraints and a reduced number of variables. We experimentally evaluated the two modeling approaches on benchmark instances from literature using various SMT-solvers. The proposed approach outperforms other existing exact approaches by solving more instances, which could not been solved yet by complete methods.

2 Rotating Workforce Scheduling

Rotating Workforce Scheduling addresses the problem of shift assignment and day-off scheduling in the specific case of cyclic schedules. The formal definition of the problem and its instances is based on the definition from [Musliu *et al.*, 2002]:

Instance:

- n: Number of employees.
- A: Set of m shifts (activities): $a_0, a_1, ..., a_{m-1}$, where a_0 represents the special "day-off" shift.
- w: Length of the schedule. Typically the value of w is set to 7 (one week). The total length of a planning period is $n \times w$ for one employee, because of the cyclicity of the schedule.
- *Cyclicity*: The required schedule is considered to be cyclic, that is, each employee works the schedule for the complete planning period starting at a specific week. For example in the second week the first employee will take the schedule of the second employee, the second employee will take the schedule of the third employee, and so one the last employee will take the schedule of the first employee.
- R: Temporal requirements matrix, an $(m-1) \times w$-matrix (because the day-off shift a_0 is excluded) where each element $r_{i,j}$ of matrix R defines the required number of employees for the corresponding shift (i) on that day of the week (j).
- *Prohibited shift sequences:* Certain shift sequences must be prevented from being assigned to the employees.
- MIN_w and MAX_w: Minimal and maximal possible length of work blocks. A work block consists of consecutive days on which an employee is working without having a day-off.
- Maximum and minimum length of periods of consecutive shifts: Vectors $MAXS_m$, $MINS_m$, where each element shows the maximum respectively minimum permitted length of periods of consecutive shifts.

Problem: Find a cyclic schedule (assignment of shifts to employees) that satisfies the requirement matrix, and all other constraints.

3 Modeling the Rotating Workforce Scheduling problem

In this paper we propose two encodings that use functions and predicates from different background theories. The first formulation is based on linear integer arithmetic, whereas the second formulation expresses RWS with fixed-size bitvectors and bit-level operations.

3.1 Linear Integer Arithmetic

For the first modeling approach the theory of linear arithmetic over the integer numbers is used. The first step is to define a set of variables to represent the workforce schedule. Therefore, we define one variable for each day of the planning period, i.e. $n \times w$ variables. Next, we have to specify the type of these variables. In order to simplify the constraint formulations we will use different integer numbers for the shifts, hence the type of our variables will be the *Integers*.

Let a schedule S be defined as a $n \times w$ matrix of variables, where n is the number of employees and w the number of work days.

$$S = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & \cdots & x_{1,w} \\ x_{2,1} & x_{2,1} & \cdots & \cdots & x_{2,w} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ x_{n,1} & x_{n,1} & \cdots & \cdots & x_{n,w} \end{pmatrix} \tag{1}$$

Furthermore, we map every shift $a_0, a_1, \ldots, a_{m-1}$ of A (see Section 2) to an integer number. Therefore, we define a function: $f : A \to \mathbb{N}$ s.t. for all $a_i \in A \setminus \{a_0\} : f(a_i) = f(a_{i-1}) * n + 1$ and $f(a_0) = 0$ (explained below).

Every variable $x_{i,j}$ of schedule S is now assigned exactly one of our calculated integer values: $\forall x_{i,j} \in S : x_{i,j} \in A'$ and $A' = \{f(a_k) : \forall a_k \in A \; (0 \leq k \leq m-1)\}$

We will now focus on formulation of the constraints using linear integer arithmetic with equalities.

Temporal requirements The main reason why we use the function f to map every shift character a_i to a natural number is that it allows us to specify the temporal requirement constraints by means of basic linear integer arithmetic. Thus, we can formally define the temporal requirements $T(j)$ for a specific day j of the week as follows:

$$T(j) = \sum_{i=1}^{m-1} r_{i,j} * f(a_i) \tag{2}$$

We can now formulate the temporal requirement constraints as an equation that always has to hold:

$$\sum_{i=1}^{n} x_{i,j} = T(j), \text{ where} (1 \leq j \leq w) \tag{3}$$

Since we defined the calculation $f(a_i)$ as the arithmetic operation $f(a_{i-1}) * n + 1$, it is ensured that the required value for $T(j)$ can only be obtained if the correct amount of staff is assigned to the corresponding shifts.

Shift Block Length To address the block length issue, we have to think of our schedule as a one dimensional sequence of work days instead of the matrix we defined before. This ensures that the last day of each week is adjacent to the first day of the next week. Additionally, we must not forget that in a cyclic schedule the last day of the last week is also adjacent to the first day of the first week. With that in mind we define the block length constraints by considering every variable to be the first in a new block. The neighbors of each of these possibly block-starting variables are then constrained: The block-starting variable itself has a certain shift value, then all following variables of the same block are restricted to the according shift value. For the following definitions we address every variable $x_{i,j}$ by its absolute position in the one dimensional representation of the schedule. Therefore, we omit the second index j as i will range from 1 to $n \times w$.

Minimum Shift Block Length: For every x_i, if $x_i = a_j$ (where a_j defines the shift to be constrained) and $x_{i-1} \neq a_j$, then $x_{i+1} = a_j \wedge x_{i+2} = a_j \wedge \ldots \wedge x_{i+min-1} = a_j$. We note that we have to avoid the cases when the subindex of x is out of bounds.

Maximum Shift Block Length: For every x_i, if $x_i = a_j \wedge x_{i-1} \neq a_j \wedge x_{i+1} = a_j \wedge x_{i+2} = a_j \wedge \ldots \wedge x_{i+max-1} = a_j$, then $x_{i+max} \neq a_j$.

Note that the formulation of the shift block constraint must also be applied to the day-off shift a_0 to restrict the minimum and maximum number of consecutive days off. We can describe the constraints for the work block length very similarly to those for the shift blocks. The only difference is that we do not care about the shifts the block consists of, unless they are assigned the special "day-off" shift a_0. Due to this similarity and space limitation the formal definition for work block length is not given in this paper (this definition can be found in [Erkinger, 2013]).

Illegal Shift Sequences Safety concerns and legal reasons make it necessary to forbid certain shift sequences from being assigned to the employees. This means that if on a day x_i an employee works in shift a_j, the employee is not allowed to work in shift a_k the day after. For every x_i it has to hold that if $x_i = a_j$, then $x_{i+1} \neq a_k$ $(k \neq j)$, iff $a_j \rightarrow a_k$ is forbidden. This assertion can be easily adapted to restrict illegal sequences of three shifts.

3.2 Bitvector Theory

The formulation of the problem in bitvector theory is more complex than the formulation using linear integer arithmetic. By definition a bitvector in the SMT context may be of arbitrary length. Therefore we propose using bitvectors that are the size of the complete schedule for the problem. Thereby each bit of a bitvector represents one day in the schedule. With that in mind, we use one bitvector per shift and one extra bitvector to represent the days-off. We will call the bitvector of a shift a *shiftvector*. A shift is assigned to a specific work day iff the according bit in the shiftvector is set to 1.

A schedule S is represented by m bitvectors $\textit{shift}_0, \textit{shift}_1, \ldots \textit{shift}_{m-1}$, where $\textit{shift}_0$ represents the day-off shift. Each bitvector $\textit{shift}_i$ $(0 \leq i \leq m-1)$ is of size $n \times w$ where n represents the number of employees and w the number of work days.

The subsequent sections discuss the constraints of rotating workforce scheduling using the bitvector encoding.

Shift Assignment The shift (including day-off) assignment constraint ensures that exactly one shift is assigned to each day of the schedule. This is accomplished in two steps: we have to make sure that there can be no more than one shift per day and employee and that there must be at least one shift assignment per day and employee.

For the first part of the constraint we use the bitwise *AND* operation. We have to apply this operation to every combination of two shiftvectors and assert that the result of the combination is a bitvector of the same length as the original vectors containing only zero-bits (0).

$$\begin{array}{ccccccccc l}
\& & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & (\textit{shift}_1) \\
 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & (\textit{shift}_2) \\
\hline
= & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\hline
 & \multicolumn{9}{c}{\text{false}}
\end{array} \qquad (4)$$

The example above shows a violation of this constraint because on two days both $\textit{shift}_1$ and $\textit{shift}_2$ have the according bits set to 1 meaning the employee would be assigned both shifts on these days.

To ensure correctness we also have to rule out that no shift is assigned to a specific day. Recall that this also includes the day-off shift ($\textit{shift}_0$). The second assertion addresses this issue by using the bitwise *OR* operation. If the result of the bitwise *OR* applied to all shiftvectors is equal to the bitvector containing only 1-bits, then at least one shift is assigned to each work day.

Temporal requirements Instead of only regarding the structure of our shiftvectors as having the length of the whole schedule ($n \times w$), we have to think about the schedule as a multidimensional array, consisting of n arrays with size w in order to restrict the number of employees assigned to each shift on a specific day. Note that a shiftvector is read from right to left, meaning that the first week of the schedule starts with the least significant bit.

$$\begin{array}{c}
\underbrace{0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1}_{\textit{week 2}} \quad \underbrace{0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1}_{\textit{week 1}} \\
\Downarrow \\
\begin{array}{ccccccc l}
0 & 1 & 1 & 1 & 0 & 0 & 1 & \textit{(week 1)} \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & \textit{(week 2)}
\end{array}
\end{array} \qquad (5)$$

Because of the cyclicity of the schedule, every week also stands for the current shift schedule of one employee. Hence, every column of our multidimensional array represents the shift plan for all employees on that day of the week. In the formulated constraint the number of 1-bits in each column must be equal to the number of employees that should be assigned to the corresponding shift on that day. Therefore, we first have to create a temporary bitvector by extracting the correlating bits of the original shiftvector and concatenating them to a new bitvector that represents that day.

$$\begin{array}{c}
0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1 \\
\Downarrow \\
\begin{array}{cc l}
1 & 1 & \textit{(Monday)} \\
0 & 0 & \textit{(Tuesday)} \\
1 & 0 & \textit{(Wednesday)} \\
 & & \ldots
\end{array}
\end{array} \qquad (6)$$

The *SMT-LIB* input language provides an `extract` operation to select certain bits within a bitvector. These bits can be concatenated with the help of the `concat` operator.

In the next step we count all bits in the bitvector that are set to 1. To accomplish that, we calculate the *Hamming Weight* of our vector. In the following a short definition of the *Hamming Weight* is given:

"The Hamming distance $d(u, v)$ between two words u and v, of the same length, is equal to the number of symbol places in which the words differ from one another. If u and v are of finite length n then their Hamming distance is finite since $d(u, v) \leq n$."[Daintith *et al.*, 2004]. Introduced by *Richard Hamming* in [Hamming, 1950], the Hamming distance is used in telecommunication for error detection and error correction. The *Hamming Weight* is simply defined as the Hamming distance of a word u and the *zero-word*. Hence, it represents the number of symbols which are different from the *zero-symbol*.

The original algorithm to calculate the Hamming Weight has a runtime that depends on the length of the input word, because it checks if every single symbol of the word differs from the zero-symbol. This approach is not ideal for our use, as it cannot be formulated in a declarative way and therefore is not suitable for us in the constraint formulation.

A better way to calculate the Hamming Weight is provided by the algorithm developed by Peter Wegner in 1960 [Wegner, 1960], which is based on the fact that if the bitwise AND operation is applied to a binary number N and $N-1$, then the resulting number has one 1-bit less than the original number N. In other words, the least significant 1-bit is eliminated.

```
input  : A bitvector $v$
output: The Hamming weight of bitvector $v$
1 while v $\neq 0$ do
2     weight = weight + 1;
3     v = v & (v- 1);
4 end
5 return weight
```

Algorithm 1: Wegner's Method for Hamming Weight

Wegner's method offers the advantage that it can be formulated declaratively using recursion, because the number of iterations (recursive calls) is equal to the number of set bits in the input vector. Formally the number of selected bits n in a bitvector v can be calculated by the recursive application of $v = v \,\&\, (v-1)$ until $v = 0$.

To use this method in our encoding we define a helper function (bitcount) that takes a bitvector v as input and returns the result of the operation $v = v \,\&\, (v-1)$.

To make sure that a certain bitvector has exactly n bits that are set to 1, we define one assertion which guarantees that after $n-1$ applications of the bitcount function the bitvector is **not** equal to 0 and another assertion that ensures the vector is equal to 0 after n calls of the bitcount function. These assertions are defined for the previously created bitvectors of each work day.

Maximum Shift Block Length Shift blocks are a sequence of 1-bits in a shiftvector. The length of this sequence needs to be constrained to prevent an employee from working an illegal number of consecutive work days of the same shift. To accomplish that, we need to count the consecutive 1-bits of a shiftvector.

We can find two consecutive bits in a bit vector v, by shifting the vector one bit to the right $v >> 1$ and combine it with the original vector using the AND operation $v \,\&\, (v >> 1)$. Then the bit v_i is set to 1 iff v_{i+1} is set to 1 in the original vector. Hence, with every application of this operation the left most bit of a block is replaced by a 0.

To find n consecutive bits we can expand our equation to this generalized one:

$$0 = v \,\&\, (v >> 1) \,\&\, (v >> 2) \,\&\, \ldots \,\&\, (v >> n) \tag{7}$$

$$\begin{array}{c|cccccccccccc|l}
\& & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & (v) \\
 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & (v >> 1) \\
\hline
\& & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & (v') \\
 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & (v >> 2) \\
\hline
\& & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & (v') \\
 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & (v >> 3) \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (v')
\end{array} \tag{8}$$

Thus, we can use this kind of algorithm to restrict the maximum length (max) of a shift block, since the vector has to be equal to 0 after at most max iterations of $v' = v' \,\&\, (v >> i)$ with $1 \leq i \leq max$, v being the original shiftvector and v' being the resulting modified shiftvector v (see Example 8 for a maximum of 3).

Another problem we have yet to deal with is that of cyclicity. As we are designing a rotating schedule, the least and the most significant bit (*lsb/msb*) of each shiftvector are adjacent, which is important for the shift block constraints. To incorporate that, we simply concatenate every shiftvector with itself so that the *lsb* and the *msb* become adjacent in the middle of the new vector. Afterwards, we apply the consecutive bitcount method to this double-size bitvector.

$$\begin{array}{c}
\mathbf{1} \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad \mathbf{1} \qquad (v) \\
\Downarrow (v \cup v) \\
\mathbf{1} \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad \mathbf{1} \quad \mathbf{1} \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad \mathbf{1}
\end{array} \tag{9}$$

Minimum Shift Block Length The next step is to ensure the minimum length of a shift block. This constraint is more difficult to formulate declaratively than the maximum shift block constraint. If we tried the same approach as for the maximum shift block length by applying the consecutive bitcount algorithm min times in order to assert that the vector equals to 0, this would clearly be wrong. As min only represents the minimum block length, longer sequences of consecutive bits are still allowed to exist. Hence, the resulting shiftvector not necessarily equals to 0.

The method we propose to ensure the minimum shift block length consists of three steps:

1. **Delete illegal blocks**: Apply $min-1$ iterations of the consecutive bit count method to the shiftvector, where min is the minimum block length. After $min-1$ applications at least one 1-bit of each legal block is still present, while all blocks of illegal size have been replaced by 0-bits.

2. **Restore blocks**: By iteratively shifting the vector from (1) to the left $min - 1$ times and applying the OR operation to the shifted vector and the one from the previous iteration, the length of all blocks is restored (except the already deleted illegal blocks).
3. **Verify**: In the last step we have to compare the bitvector from (2) with the original shiftvector. These two vectors are equal iff the original vector did not contain any illegal blocks in the first place, because otherwise they would have been deleted in step (1).

```
input  : A bitvector $v$
input  : The minimum block length $min$
output: A Boolean indicating that the min block length holds
        for $v$ or not
$v_2 = v$
for $i = 1$ to $min - 1$ do
    $v_2 = v_2 \ \& \ (v_2 >> 1)$
end
for $i = 1$ to $min - 1$ do
    $v_2 = v_2 \mid (v_2 << 1)$
end
return $v == v_2$
```

Algorithm 2: Minimum Shift Block Length

For better illustration of how the algorithm works, consider the following example. Take a shiftvector $shift_1$ and check if the minimal block length $min = 3$ holds for all shift blocks of this vector.

$$0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \qquad (10)$$

We will start by applying the consecutive bitcount method $min - 1$ times:

&	0	1	1	1	0	1	0	1	1	1	0	
	0	0	1	1	1	0	1	0	1	1	1	(>> 1)
&	0	0	1	1	0	0	0	0	1	1	0	
	0	0	0	1	1	0	0	0	0	1	1	(>> 1)
	0	0	0	1	0	0	0	0	0	1	0	

(11)

The example above shows that with every iteration of the consecutive bitcount method, the left most bit of every 1-block is replaced by a zero. Hence, after $min - 1 = 2$ iterations at least one set bit of each legal block has not been replaced, whereas the illegal blocks – blocks with a length less than min – have vanished completely.

As a next step we will restore the original shiftvector by shifting the vector back to the left $min-1$ times and applying the bitwise OR operation to every iteration (see 12). This will ensure that every 1-bit of a legal block that has been replaced by the consecutive bitcount method is being restored, while the blocks with illegal length are not.

|	0	0	0	1	0	0	0	0	0	1	0	
	0	0	1	0	0	0	0	0	1	0	0	(<< 1)
|	0	0	1	1	0	0	0	0	1	1	0	
	0	1	1	0	0	0	0	1	1	0	0	(<< 1)
	0	1	1	1	0	0	0	1	1	1	0	

(12)

The resulting vector is now compared to the original shift vector. The constraint is successfully fulfilled iff the two vectors are equal.

=	0	1	1	1	0	1	0	1	1	1	0
	0	1	1	1	0	**0**	0	1	1	1	0
									false		

(13)

So far we have left out the fact that our algorithm does not consider the cyclicity of the schedule, meaning that the first and the last bit of a shiftvector are adjacent. To overcome this issue, we have to refine our algorithm in a way that incorporates rotating schedules. Suppose we have the following shift vector:

$$1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \qquad (14)$$

If we constrain the minimal block length to 3, the shiftvector will be legal, because we have two blocks of length 3, one being in the middle of the vector and one that consists of the first and the last two bits of the vector. Before we can make use of our algorithm, we therefore have to concatenate this split block, since otherwise the algorithm would falsely classify two blocks as illegal, one each at the beginning and the end of the vector. Thus, by moving the illegal block at the end in front of the most significant bit, we merge these blocks together and solve this issue.

We start out by creating a vector that only contains the last 1-block of the original shiftvector.

¬	1	0	0	1	1	1	0	1	1	(v)
&	0	1	1	0	0	0	1	0	0	(¬v)
	0	1	1	0	0	0	0	1	1	(¬v - 1)
⊕	0	1	1	0	0	0	0	0	0	(v')
	0	1	1	0	0	0	1	0	0	(¬v)
2-complement	0	0	0	0	0	0	1	0	0	(v')
	1	1	1	1	1	1	1	0	0	(v')
¬	0	0	0	0	0	0	0	1	1	(v')

(15)

Example 15 shows how the vector can be modified, so that only the least significant 1-block is present. We could now concatenate this vector with the original shift vector in order to merge the split block. However, there is still a problem. Although we have concatenated the two vectors and thus considered the split block as a legal block, we still have two bits at the end of the vector, that form an illegal block. Therefore, before concatenating the two vectors, we apply the XOR operation to them and then concatenate the vector from 15 with the resulting vector of that operation.

⊕	1	0	0	1	1	1	0	1	1
	0	0	0	0	0	0	0	1	1
	1	0	0	1	1	1	0	0	0

$$\Downarrow$$

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0$$

(16)

The obtained vector represents the correct non-cyclic shiftvector. The minimum shift block length algorithm (Algorithm 2) applied to this vector will indicate whether the shiftvector contains only legal blocks or not.

Work Block Length: The work block constraints are very similar to the shift block constraints. The only difference is that a work block is a sequence of consecutive days where an employee has to work regardless of what shift he/she is assigned to.

We could get a vector encompassing all the work blocks by simply applying the bitwise OR on all shiftvectors, except the special “day-off” vector. Another way to obtain that vector is to negate the “day-off” vector, i.e. replace every 0 by 1 and every 1 by 0. The 1-bits in the resulting vector then represent all work-days, whereas the 0s stand for the days off. The easiest way to negate a bitvector is to use the NOT operation .

$$\begin{array}{ccccccccc l} \neg & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & (shift_0) \\ \hline & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & \neg(shift_0) \end{array} \qquad (17)$$

We now use the same constraint formulations as for the shift block length on this new vector to restrict the length of consecutive work days.

Illegal shift constellations Certain shift constellations are considered illegal, because every employee needs a fixed amount of resting time between two workdays. Therefore, it is not allowed for example, to work the day-shift after working the night-shift.

$$\begin{array}{cccccccc l} 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & \text{(Night shift)} \\ \mathbf{1} & 0 & \mathbf{1} & 1 & 1 & 0 & 0 & 0 & \text{(Day shift)} \end{array} \qquad (18)$$

These constellations can be forbidden with the help of bit rotation of a shiftvector. Considering the example above, to prevent the shift constellation *Night* $\rightarrow$ *Day* we have to rotate the shiftvector of the night shift N by 1 to the right. We can now AND this vector with the shiftvector of the day shift D and check if they have no bits in common that are set to 1. Subsequently, for any position i, $D_i \neq N_i$ has to hold. Hence, the resulting bitvector must be 0.

$$\begin{array}{c cccccccc l} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & \text{(Night shift)} \\ & \mathbf{1} & 0 & \mathbf{1} & 1 & 1 & 0 & 0 & 0 & \text{(Day shift)} \\ & & & & & & \Downarrow & & & \\ \circlearrowright & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & \text{(Night shift)} \\ \hline \& & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & \text{(rotated Night shift)} \\ & \mathbf{1} & 0 & \mathbf{1} & 1 & 1 & 0 & 0 & 0 & \text{(Day shift)} \\ \hline = & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\ \hline & & & & & \multicolumn{4}{l}{\text{false}} & \end{array} \qquad (19)$$

The example also illustrates why it is important to use bit rotation instead of a logical shift right operation. If we had shifted the night shift vector to the right, we would have missed the fact that the least and most significant bit of each vector are adjacent.

The problem definition also allows illegal shift sequences of length 3 to be specified. The algorithm for detecting such illegal sequences is still very similar, except that we have to take three shiftvectors into consideration. A more detailed explanation can be found in [Erkinger, 2013].

4 Experimental Results

In this section we investigate the performance of our two modeling approaches on 20 benchmark instances described in [Musliu, 2005; 2006].[1] The first three problems have been used in most rotating workforce scheduling papers given in the introduction and other 17 instances were derived from real life problems of different business areas. The collection contains small, middle-size and very large instances. The number of employees (or groups) for the largest instance is 163. As the number of shifts for this problem is 4 (including day-off), the search space for this problem is 4^{163*7}.

We applied five state-of-the-art SMT-solvers (Z3 version 4.3.1 [de Moura and Bjørner, 2008], MathSAT5 version 5.2.5 (gmp 5.0.5, clang/LLVM 3.1, 64-bit) [Cimatti *et al.*, 2013], CVC4 version 1.0 (compiled with GCC version 4.2.1) [Barrett *et al.*, 2011], Yices version 2.1.0 [Dutertre and de Moura, 2006] and Boolector version 1.5.115 [Brummayer and Biere, 2009]). The solving process was performed on an Apple MacBook Pro with the following hardware details: 2.0 GHz quad-core Intel Core i7 (Sandy-Bridge) and 8 GB DDR3 RAM. The timeout for each solver and instance was set to 1000 seconds.

A comparison of the different solvers with respect to the number of found solutions for the the linear arithmetic (LIA) encoding shows that *Z3* is able to solve the most benchmark instances (*Z3* finds a solution for 15 instances). All five solvers achieve a higher solution rate with the bitvector approach than with the linear arithmetic method. *MathSAT* and *Boolector* were even able to solve 18 instances, which is three instances more than the best solver (Z3) of the LIA approach. A detailed evaluation including the exact runtimes can be found in [Erkinger, 2013].

Next, we compare the fastest performing SMT-solvers of each modeling approach to each other in order to evaluate whether one encoding is superior for solving the rotating workforce scheduling problem (Figure 1). The best performing solver of the linear arithmetic approach was *Z3*, whereas *MathSAT* provides the best results for the bitvector encoding. We can conclude from Figure 1 that the bitvector approach outperforms the linear arithmetic method on all instances and achieves significantly faster solving times. This seems to confirm the initial assumption that modeling the problem with bitvectors has a positive impact on the solvers' runtime.

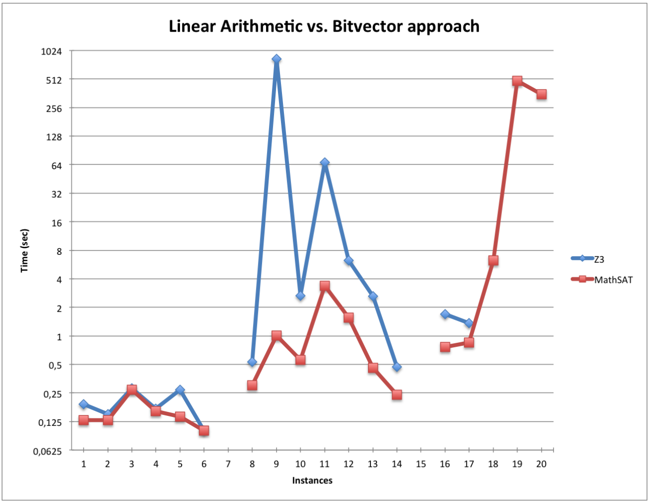

Figure 1: Comparison of the fastest solver of each approach

[1] http://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/workforceScheduling.zip

5 Comparison to other Approaches

In order to evaluate whether the approach of solving the rotating workforce scheduling problem with SMT-solvers is competitive, we have to compare the results to other solving techniques from literature. First, we compare the results of MathSAT to two other approaches ([Balakrishnan and Wong, 1990] and [Laporte and Pesant, 2004]) that have been tested on three existing instances from literature. Table 1 provides the solving times for both of these approaches and MathSAT. Although the results cannot be compared directly due to different hardware specifications, it can be observed that MathSAT solves these three instances in a short span of time and thus is competitive with the two approaches.

Instance	MathSAT 5	BW90	LP04
1 [Butler, 1978]	0,13	73,54	3,78
2 [Laporte *et al.*, 1980]	0,13	310,84	0,03
3 [Heller *et al.*, 1973]	0,27	457,98	10,26

Table 1: Comparison of solving times to previous methods

We now compare the number of solutions and the runtimes to those of the *First Class Scheduler* (FCS) proposed by [Musliu *et al.*, 2002], the min-conflicts heuristic with tabu-search algorithm (MC-T) of [Musliu, 2006] and the exact method developed by [Triska and Musliu, 2011] that uses constraint programming. MC-T solves all 20 instances, however, it is a heuristic technique and should therefore not be directly compared to the exact methods. It is more suitable to compare the SMT-technique to FCS [Musliu *et al.*, 2002] and the constraint programming approach CP-Rota [Triska and Musliu, 2011], as these two are also exact methods.

In comparison to the *First Class Scheduler*, the SMT approach solves 5 instances more than FCS. While MathSAT can find a solution to the instances 9,12,13,17,19 and 20, FCS did not finish within 1000s on these instances. On the other hand, FCS is able to solve instance 7 which is not the case for any of the tested SMT-solvers in this paper.

Comparing the constraint programming approach CP-Rota to the results of MathSAT shows that CP-Rota solves 13 out of 20 instances, meaning five less than the SMT-solver. Although MathSAT is able to find a model for the instances 10, 11, 12, 18, 19 and 20, CP-Rota finds a solution for instance 7 in return. It is indeed surprising that the other exact methods were not only able to solve instance 7, but to do so in a short time. We could not find an explanation why SMT-solvers fail to find a solution for this instance, especially as they share many similarities with constraint programming techniques. Nevertheless, MathSAT has the highest solution rate of all exact methods considered in this paper.

Although these results indicate the superiority of SMT-solving over other exact methods, this conclusion should be taken with caution since the runtimes of these techniques were not measured on the same hardware and are therefore not directly comparable. We now have a look at the detailed solving times given in Table 2.

Instance	MathSAT 5	MC-T	FCS	CP-Rota
1	0,13	0,07	0,9	0,02
2	0,13	0,07	0,4	0,02
3	0,27	0,42	1,9	0,24
4	0,16	0,11	1,7	0,03
5	0,14	0,43	3,5	0,98
6	0,1	0,08	2	0,02
7	>1000?	52,79	16,1	0,07
8	0,3	0,74	124	964
9	1,01	15,96	>1000?	19
10	0,56	0,60	9,5	>1000?
11	3,37	13,15	367	>1000?
12	1,56	1,17	>1000?	>1000?
13	0,46	0,87	>1000?	114
14	0,24	0,76	0,54	940
15	>1000?	159,04	>1000?	>1000?
16	0,76	0,54	2,44	216
17	0,85	2,16	>1000?	18
18	6,23	6,83	2,57	>1000?
19	489,32	75,83	>1000?	>1000?
20	355,21	71,38	>1000?	>1000?

Table 2: Comparison of solving times with FCS and CP-Rota

The runtimes show that the two approaches from literature achieve good results on a number of instances. This indicates that FCS and CP-Rota might achieve better results on a faster hardware and thus even outperform the MathSAT solver. However, the SMT approach is particularly fast on most instances (9, 10, 11, 12, 17, 18) that could not be solved by FCS or CP-Rota.

In conclusion, the encoding in SMT is an efficient approach to rotating workforce scheduling, which provides promising results when compared to other exact methods.

6 Conclusion

In this paper we proposed a new approach for solving the rotating workforce scheduling problem. By formulating rotating workforce scheduling as an SMT problem, we were able to use state-of-the-art solving tools (SMT-solvers) for the generation of cyclic workforce schedules. With the help of different background theories we developed two modeling approaches for the problem. Although encoding the problem with bitvectors and bit-level operations was a challenging task due to the complexity of the constraints, the assumption of the approach's superiority was indeed confirmed by the comparison of the experimental runtime results of the two modeling techniques. The competitiveness of solving the problem with SMT-solvers was verified by the comparison to other exact methods from literature. The results show that SMT-solving is a well performing alternative to these approaches, because it outperformed the other exact approaches on many instances and is able to provide a solution in a reasonable time frame for some instances where the other methods fail to do so.

Acknowledgments

The work was supported by the Austrian Science Fund (FWF): P24814-N23.

References

[Ansótegui *et al.*, 2011] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem. In *Proceedings of SARA - Symposium on Abstraction, Reformulation and Approximation*. AAAI, 2011.

[Ansótegui *et al.*, 2013] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Solving weighted csps with meta-constraints by reformulation into satisfiability modulo theories. *Constraints*, 18(2):236–268, 2013.

[Balakrishnan and Wong, 1990] Nagraj Balakrishnan and Richard T Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20(1):25–42, 1990.

[Barrett *et al.*, 2011] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[Bofill *et al.*, 2010] Miquel Bofill, Josep Suy, and Mateu Villaret. A system for solving constraint satisfaction problems with SMT. In *Theory and Applications of Satisfiability Testing – SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 300–305. Springer, 2010.

[Brummayer and Biere, 2009] Robert Brummayer and Armin Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *Lecture Notes in Computer Science*, pages 174–177. Springer, 2009.

[Butler, 1978] B. Butler. Computerized manpower scheduling. Master's thesis, University of Alberta, Canada, 1978.

[Cimatti *et al.*, 2013] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.

[Daintith *et al.*, 2004] J. Daintith, V. Illingworth, and I.C. Pyle. *Oxford Dictionary of Computing*. Oxford paperback reference. Oxford University Press, 2004.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proceedings of TACAS - Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, chapter 24, pages 337–340. Springer, 2008.

[Dutertre and de Moura, 2006] Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Technical report, SRI International, 2006.

[Erkinger, 2013] Christoph Erkinger. Rotating workforce scheduling as satisfiability modulo theories. Master's thesis, Vienna University of Technologie, May 2013.

[Falcón *et al.*, 2016] Raúl Falcón, Eva Barrena, David Canca, and Gilbert Laporte. Counting and enumerating feasible rotating schedules by means of gröbner bases. *Mathematics and Computers in Simulation*, 125:139–151, 2016.

[Hamming, 1950] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[Heller *et al.*, 1973] Nelson B Heller, J Thomas McEwen, and William W Stenzel. *Computerized scheduling of police manpower*. St. Louis Police Department, St. Louis, MO, 1973.

[Laporte and Pesant, 2004] Gilbert Laporte and Gilles Pesant. A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55(11):1208–1217, 2004.

[Laporte *et al.*, 1980] Gilbert Laporte, Yves Nobert, and Jean Biron. Rotating schedules. *European journal of operational research*, 4(1):24–30, 1980.

[Laporte, 1999] G Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, 9 1999.

[Musliu *et al.*, 2002] Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.

[Musliu, 2005] Nysret Musliu. Combination of local search strategies for rotating workforce scheduling problem. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 1529–1530, 2005.

[Musliu, 2006] Nysret Musliu. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4):309–326, 2006.

[Nieuwenhuis and Oliveras, 2006] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo theories and optimization problems. In *Proceedings of SAT - Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006.

[Triska and Musliu, 2011] Markus Triska and Nysret Musliu. A constraint programming application for rotating workforce scheduling. In *Developing Concepts in Applied Intelligence*, volume 363 of *Studies in Computational Intelligence*, pages 83–88. Springer Berlin / Heidelberg, 2011.

[Wegner, 1960] Peter Wegner. A technique for counting ones in a binary computer. *Commun. ACM*, 3(5):322, 1960.